VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Analýza vytížení nad SQL Server databází

SQL Server Database Workload Analysis

2015 Ondřej Krajčík

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Ondřej Krajčík**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Analýza vytížení nad SQL Server databází
SQL Server Database Workload Analysis

Zásady pro vypracování:

Cílem této práce je vytvořit nástroj pro analýzu souboru obsahujícího vytížení nad SQL Server databází. Vytížení bude ve formě textového souboru obsahujícího posloupnost SQL příkazů nad databázi.
V rámci analýzy se provedou tyto úkony:
1. Nalezení SQL příkazů, které se liší pouze svými parametry.
2. Oddělení hodnot parametrů SQL příkazů do samostatných seznamů.
3. Vytvoření grafu, který bude zachycovat návaznost SQL příkazů ve vytížení.
4. Uložení unikátních SQL příkazů a jejích parametrů do JSON souboru.
5. Otestování aplikace pro několik netriviálních vytížení zachycených přímo na SQL Serveru (parametrizované/neparametrizované dotazy, procedury, sp_executesql).

Seznam doporučené odborné literatury:

Podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Radim Bača, Ph.D.**

Datum zadání: 01.09.2014

Datum odevzdání: 07.05.2015

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 4. 5. 2015 Podpis: [signature]

Rád bych na tomto místě poděkoval vedoucímu této bakalářské práce, panu Ing. Radimu Bačovi, Ph.D., za jeho cenné rady, nápady a celkové vedení práce.

Abstrakt

Tato práce se zabývá analýzou vytížení Microsoft SQL Serveru z pohledu ladění výkonu databáze. Hlavní náplň práce tvoří vytvořená aplikace, která nabídne uživateli rychlý přehled nad vytížením databázového serveru. První část práce se zabývá využíváním nástojů SQL Serveru, jejich výhodami a nevýhodami. Druhá část je zaměřena na popis vytvořené aplikace a její testování.

Klíčová slova: Analýza, SQL Server, databáze, vytížení

Abstract

This study deal with analysis of Microsoft SQL Server workload in terms of performance improvements. The main purpose of the study is build application which provide fast overview over database server workload to user. The first part of the study deal with utilization of SQL Server tools, their advantages and disadvantages. Second part deal with description and testing of build application.

Keywords: analysis, SQL Server, workload, database

Seznam použitých zkratek a symbolů

SQL	–	Structured Query Language
SŘBD	–	Systémem řízení báze dat
XML	–	Extensible Markup Language
MSAGL	–	Microsoft Automatic Graph Layout
DBA	–	Database administrator
IS	–	Informační systém
GUI	–	Graphical User Interface

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Výkon databázových serverů patří k významným aspektům informačních systémů. Tyto databázové servery tvoří uložiště dat statisícům zařízení a jejich výkon má přímo úměrný vliv na odezvu či počet zařízení, které je server schopen obsloužit. Výkon lze ovlivnit na úrovni hardwarové a softwarové. Investice do výkonějšího hardwaru však nemusí být vždy tou správnou cestou. Výkonější hardware sice vede ke zvýšení výkonu databázového serveru, ale často databázové servery plýtvají výkonem zcela zbytečně z důvodu špatného fyzického návrhu databáze či špatné konfigurace. Pokud máme moderní hardware, který nezvládá zpracovávat všechny požadavky, je na místě zabývat se fyzickým návrhem a konfigurací. V opačném případě, pokud máme dobrý fyzický návrh a dobrou konfiguraci, měli by jsme uvažovat nad koupí výkonějšího hardwaru. Je tedy vždycky na zvážení zda je výhodnější investovat do výkonějšího hardwaru nebo se zabývat laděním databáze.

Cílem mé práce je vytvoření aplikace analyzující provoz, respektive zachycení vytížení nad Microsoft Sql Serverem. Hlavní částí analýzy je nalezení SQL příkazů, které se liší pouze svými parametry, tyto parametry oddělit do samostatných seznamů a vytvoření *grafu návazností SQL příkazů* reprezentujícího vytížení. Analýzu vytížení lze využít k ladění databázového serveru či samotné databáze což vede ke zvýšení výkonu na softwarové úrovni.

První část práce se zabývá popisem dostupných nástrojů k zachycení vytížení Microsoft Sql Serveru, popisu nástrojů a následnou analýzou.

Druhá část je zaměřena na popis funkcí vytvořené aplikace, vizí do budoucna, o které by mohla být aplikace rozšířena. Jsou zde také zmíněny použité knihovny a testování aplikace na reálných vytíženích odlišných velikostí obsahující různé typy SQL příkazů.

V závěru shrnuju výsledky své práce, získané zkušenosti a poznatky týkající jak Sql Serveru tak vytvořené aplikace. Dále se zde zabývám problematikou licencí knihoven, které jsou v aplikaci využity. V kapitole také nechybí zmíněny rozšíření aplikace do budoucna.

2 Teoretická část

2.1 Monitorování databáze

Jeden z hlavních důvodů pro sledování výkonu databázového serveru je řešení problémů. Například může nastat situace, kdy mají uživatelé potíže s připojením k serveru. Pak je vhodné daný server monitorovat a zjistit přesně za jakých okolností problém nastává. Cílem je vystopovat problémy pomocí dostupných nástrojů pro sledování serveru a poté je účinně vyřešit.

Dalším častým důvodem pro monitorování serveru je zlepšení jeho výkonu. Je potřeba dosáhnout optimálního výkonu, aby uživatelé čekali na výsledek SQL příkazu co nejkratší dobu a aby server zvládl zároveň zpracovat co nejvíce souběžných požadavků. [1].

2.2 Administrace databázového serveru

Správce databáze (DBA) je odpovědný za výkon, integritu a bezpečnost v databázi. Dodatečné požadavky většinou zahrnují plánování, vývoj a řešení problémů.

DBA musí ctít následující zásady:

- Data zůstávají konzistentní napříč celou databází.
- Data jsou jasně definovaná.
- Uživatelé přistupují k datům současně, v podobě, která vyhovuje jejich potřebám.
- Existuje ustanovení pro zabezpečení dat a obnovení kontroly (všechny údaje jsou zjistitelné v případě nouze).
- Práce správce databáze (DBA) se liší v závislosti na povaze zaměstnávající organizace a úrovni odpovědnosti v souvislosti s pozicí. Práce může být čistě údržba nebo může také zahrnovat vývoj databází.

Typické odpovědnosti zahrnují některé nebo všechny z následujících akcí:

- Sledování přístupu uživatelů a bezpečnosti.
- Monitorování výkonu a správu parametrů k zajištění rychlé odezvy.
- Plánování kapacit.
- Uvedení do provozu a instalací nových aplikací.
- Instalace a testování nové verze systému pro správu databází (DBMS).
- Zajišťování bezpečnosti proti rostoucí kybernetické kriminalitě.

- Spolupráce s IT manažery projektu, programátory a webovými vývojáři.
- Zajištění ukládání, archivace, zálohování a obnovení.
- Udržování datových standardů, včetně dodržování zákona o ochraně dat.
- Vývoj, správa a testování zálohování a obnovy.
- Kontrola oprávnění přístupu uživatelů.
- Psaní dokumentace k databázi, včetně data standardů, postupů a definice datového slovníku.

[3].

2.3 Zachycení a analýza vytížení

Každý SŘDB poskytuje nástroje pro správu databázového serveru. Avšak je už na každém SŘDB, jaké množství nástrojů a jaké možnosti administrátorovi nabízí.

Pokud se budeme zabývat SŘDB Sql Server a Oracle, pak lze říct, že oba nabízejí kvalitní a propracované nástroje pro zachycení a analýzu vytížení nad databází. Podle webu DB-ENGINES [5] Sql Server a Oracle obsazují první tři místa v žebříčku populárních SŘDB spolu s MySQL

Každý SŘDB se liší v použití těchto nástrojů a možnostech analyzování vytížení. Cíl však mají všechny stejný. Popíšeme si základní možnosti databázových systémů.

2.4 Obecný popis možnosti databázových systémů

Administrátor si nejdříve musí dát pozor a rozlišovat mezi zachytáváním vytížení na straně klienta a zachytáváním na straně serveru. Zachytávání na straně klienta se nedoporučuje z důvodu zpomalení serveru. Každý SQL příkaz zaslaný na databázový server se vykoná a následně se pošle do nástroje, kde je vytížení zachytáváno. Pokud je server vytížen, stává se, že se nepodaří odchytit celý provoz. To je způsobeno tím, že databázový server už nestíhá zasílat všechny příkazy do našeho nástroje. Z těchto důvodů se doporučuje používat výhradně zachytávání na straně serveru. Zkušený administrátor by se měl zachytávání na straně klienta zcela vyhnout.

2.4.1 Popis nástrojů v Sql Serveru

Zásadní součást administrace serveru představuje monitorování výkonu, sledovaní aktivity uživatelů a řešení problémů. Microsoft SQL Server nabízí několik nástrojů, které lze pro ty to účely použít. Nástroj Sledování výkonu tvoří standardní součást systému Windows Server, určenou přímo k monitorování serverů. V tomto nástroji je možné sledovat tzv. čítače (counters). Tyto čítače umožňují sledovat mnoho různých serverových zdrojů a různé aktivity serveru. SQL Server Profiler, analytický a profilovací nástroj, umožňuje sledovat události na serveru (na straně klienta) v reálném čase. Existují však i další nástroje a zdroje v podobě uložených procedur a protokolů SQL Serveru. [1]

2.4.2 Sledování výkonu (Performance Monitor)

Nástroj *Sledování výkonu* představuje jeden z nejpoužívanějších nástrojů pro sledování výkonu SQL Serveru. Nástroj *Sledování výkonu* zobrazuje pro zvolenou množinu výkonových parametrů statistiky v podobě grafů. Těmto výkonovým parametrům se říká *čítače (counters)*.

Po nainstalování SQL Serveru se do nástroje *Sledování výkonu* vloží sada měřičů, které slouží pro sledování výkonu SQL Serveru. Tyto měřiče je rovněž možné aktualizovat po nainstalování služeb a dodatků SQL Serveru. Například po nakonfigurování replikace se přidá do nástroje *SQL Server Management Studio* komponenta *Replication Monitor*. V nástroji *Sledování výkonu* rovněž dojde k aktualizaci a přidá se do něj množina objektů a čítačů, určených ke sledování výkonu replikace.

Nástroj *Sledování výkonu* zobrazuje grafy pro různé zadané parametry. Lze zadat interval obnovy grafů, který má výchozí hodnotu tři sekundy. Nástroj Sledování výkonu poskytuje nejcennější informace v situaci, kdy se zaznamenávají informace do protokolu a jsou nakonfigurována upozornění. Tato upozornění zasílají zprávy v případě uskutečnění určité události nebo při dosažení určitého prahu. Jako příklad lze uvést okamžik, kdy začne docházet volné místo v databázovém protokolu. [1].

Mezi nejpoužívanější čítače se podle webu Database Journal [2] řadí:

- **SQLServer: Buffer Manager: Buffer cache hit ratio.**
  Čítač reprezentuje, jak často SQL Server nalezne potřebné stránky ve vyrovnávací paměti. Čím vyšší číslo, tím víc dat bylo nalezeno ve vyrovnávací paměti a data se nemusela načítat z disku. Ideální hodnota je 100, což znamená, že 100 % všech dotazovaných dat bylo nalezeno ve vyrovnávací paměti. Nízka hodnota tohoto čítače indikuje problém. Problémem je potřeba se zabývat, pátrat po příčině a efektivně ji odstranit.
  Doporučená hodnota: > 90

- **SQLServer: Buffer Manager: Page life expectancy.**
  Čítač reprezentuje jak dlouho zůstůstavají stránky v paměti. Tato hodnota je vyjádřena v sekundách. Čím větší číslo, tím pravděpodobněji SQL Server nebude potřebovat načítat stránky z disku při vyhodnocování SQL příkazů. Čítač by se měl průběžně sledovat a určit jaká hodnota je pro daný databázový server obvyklá.
  Doporučená hodnota: < 300

- **SQLServer: SQL Statistics: Batch Requests/Sec.**
  Počet SQL příkazů, které SQL Server příjme za sekundu. Jedná se o indikátor aktivity na SQL Serveru. Vyšší číslo znamená větší provoz. Neexistuje univerzální číslo, které by napovídalo, že SQL Server již provoz nezvládá. Na každém SQL Serveru je potřeba vypozorovat v jakých hodnotách se čítač pohybuje při běžném provozu.

- **SQLServer: SQL Statistics: SQL Compilations/Sec.**
  Počet kompilací plánu vykonávání SQL příkazů za sekundu. Kompilace vykonávání plánu SQL příkazů je drahá operace. Tento čítač lze porovnávat s čítačem *Batch*

Requests/Sec k indikaci zda kompilace namají příliš nepříznivý vliv na výkon. Pokud podělíme čítač Batch Requests čítačem SQL Compilations, dostaneme poměr vykonaných SQL příkazů ku kompilacím.
Doporučená hodnota: 10 příkazů ku jedné kompilaci

- **SQLServer: General Statistics: User Connections.**
  Počet uživatelů, kteří jsou k databázi připojeni. Hodnota je individuální a proto u totoho čítače je opět vhodné průběžné sledování, aby jsme si stanovili, jaké číslo je pro daný server obvyklé.

Další často používané čítače včetně doporučených hodnot lze nalézt v *Praktickém manuálu k administrasi SQL Serveru* od Radima Bači. [6]

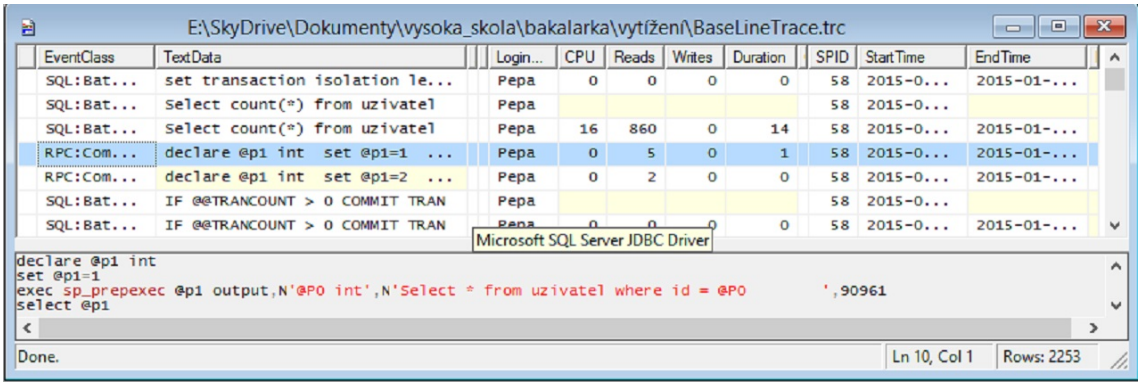
2.4.3 Sql Profiler

Ať už je potřeba sledovat aktivitu uživatelů, řešit problémy s připojením nebo optimalizovat SQL Server, nástroj SQL Server Profiler představuje jednu z nejlepších dostupných možností, alespoň do doby než bude k dispozici systém Extended Events. Systém Extended Events začíná nahrazovat SQL Server Profiler a umí trasovat i uložené procedury. Nástroj Profiler umožňuje trasovat události, k nimž v SQL Serveru dochází. Události, které lze prostřednictvím nástroje Profiler sledovat, se podobají čítačům, jež lze sledovat v nástroji *Sledování výkonu (Performance Monitor)*. Události jsou organizované do skupin zvaných *třídy událostí (event classes)*. Ve všech dostupných třídách událostí lze sledovat jednu či více událostí. Síla nástroje Profiler spočívá v jeho pokročilých funkcích a rozsáhlých možnostech vlastního nastavení.

Při analýze dat lze zaznamenávat a přehrávat trasované události. Právě v této oblasti nástroj Profiler exceluje. Lze jej použít k následujícím činnostem:

- Získání informací, které pomohou odhalit pomalé SQL příkazy a uričit, co tyto SQL příkazy zpomaluje.
- Procházení příkazů krok za krokem, nalezení příčiny problému na testovacím serveru a odhalit příčinu problému.
- Trasovat sadu příkazů, které způsobují potíže, a pak si data z trasování přehrát na testovacím serveru a odhalit příčinu problému.
- Ohalení příčiny uváznutí (deadlocks) pomocí informací z trasování.
- Sledování aktivity uživatelů a aplikací, odhalení akce, které spotřebovávají procesorový čas, nebo SQL příkazy, které se zpracovávají hodně dlouho.

[1]
SQL Profiler nabízí možnost filtrování a to na několika úrovních. První z nich je filtrování podle **kategorie událostí**. Těchto kategorií nabízí SQL Profiler spoustu. Mezi tyto kategorie patří například:



Obrázek 1: Náhled souboru s vytížením v Sql Profileru

- **Security Audit** - Změny oprávnění uživatelů.
- **Sessions** - Započetí nových session.
- **Stored Procedures** - Uložené procedury.
- **TSQL** - DDL, DML a DQL příkazy, které se spustí na serveru.

Ke každé kategorii lze zvolit atributy, které bude vytížení obsahovat. Další důležitá možnost nastavení se skrývá pod tlačítkem *"Column Filters"*, kde lze vyfiltrovat data podle libovolného atributu. Například si lze vyfiltrovat:

- Provoz určitého uživatele.
- SQL příkazy s vysokým využitím procesorového času.
- SQL příkazy s vysokým počtem logických přístupů na disk.
- SQL příkazy s dlouhým časem běhu.

U tohoto kroku je vhodné se opravdu zamyslet a nastavit filtry pouze pro zachytávání žádoucího provozu. Nežádoucí provoz zabírá zbytečné místo a komplikuje analýzu.

Poznámka 2.1 Některé atributy jsou pro import souboru povinné. Detailně popsáno v podkapitole Import 3.6.2.

Postup zachycení vytížení lze nalézt v *Praktickém manuálu k administraci SQL Serveru* od Radima Bači [6].

Zachycené vytížení je uloženo v *binárním souboru s vytížením*. Tento soubor je možné otevřít v nástroji SQL Server Profiler. V textovém editoru je *binární soubor s vytížením* nečitelný.

Na obrázku 1 lze vidět *soubor s vytížením* zobrazený v nástroji SQL Profiler. *Soubor s vytížením* obsahuje SQL příkazy zaslané na databázi a lze z něho vyčíst informace jako jsou:

- Čas začátku vykonávání SQL příkazu.
- Čas dokončení zpracování SQL příkazu.
- Doba celého běhu procesu.

Se zmíněnými atributy vytvořená aplikace pracuje. Mimo jiné *soubor s vytížením* obsahuje také informace:

- Jméno uživatele, který SQL příkaz na databázi zaslal.
- Procesorový čas.
- Počet logických čtení z disku.

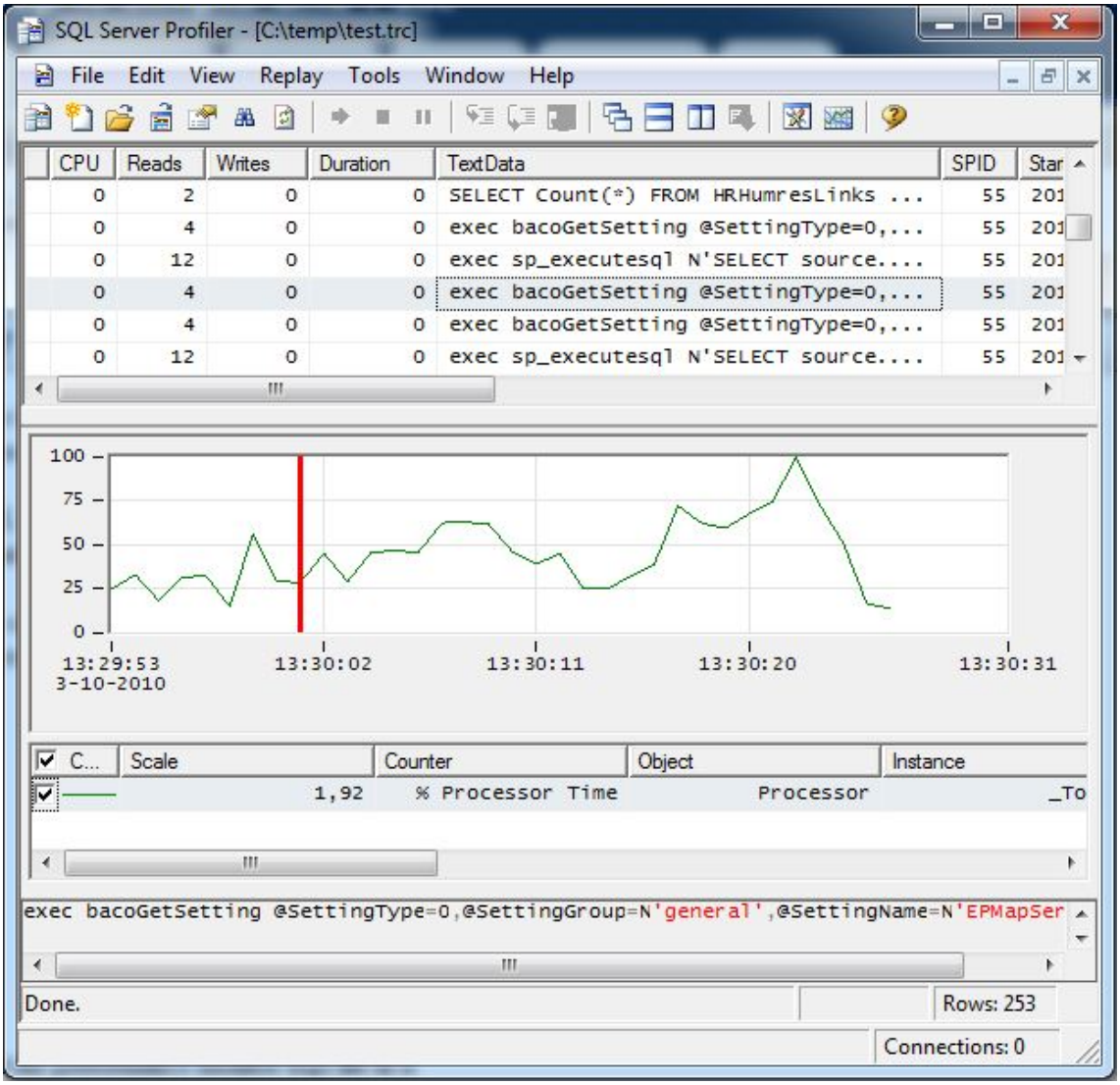
Již na první pohled lze jednoduše odhalit adepty na zlepšení fyzického návrhu a to dle sloupce *Reads* (počet logických čtení z disku). Příliš velké číslo databázi nesvědčí, protože z disku není možné přečíst informaci z jednoho místa, ale musí se přeskakovat na několik míst. V ideálním případě by měly být všechny hodnoty rovny jedné. V praxi se snažíme snížit toto číslo na minimum. U vysokého čísla Reads je potřeba prozkoumat plán vykonávání SQL příkazu a zvážit, zda není vhodné vytvořit nějaký index, pohled nebo jiné řešení. Takový zásah může způsobit zhoršení jiných operací a je potřeba si takový krok dobře rozmyslet a nejlépe odzkoušet na vytížení, zda došlo ke zlepšení.

Sql Profiler nabízí možnost exportu *souboru s vytížením* do několika formátů. První formát je *binární soubor vytížením* s příponout .trc. Tento *binární soubor s vytížením* není čitelný v textovém editoru. Pro zobrazení lze využít SQL Server Profiler. Druhý podporovaný formát pro export je *XML soubor s vytížením*. *XML soubor s vytížením* lze použít pro import do vytvořené aplikace. Třetí formát exportu je přímo do tabulky databáze SQL Serveru. Třetí variantou se tato práce nezabývá.

Poznámka 2.2 Takto vyexportovaný *XML soubor s vytížením* lze použít jako vstup do vytvořené aplikace

SQL Server Profiler umožňuje do odchyceného *souboru s vytížením* přidat data z nástroje Performance Monitoru. SQL Profiler nabízí zobrazení *souboru s vytížením* včetně požadovaných čítačů z Performance Monitoru. Skrz *soubor s vytížením* se lze proklikávat přes jednotlivé SQL příkazy a současně můžeme vidět na grafu jakých hodnot nabývaly čítače v době zaslání zvoleného příkazu. Tímto způsobem se dají velmi efektivně odhalit problémové části.

Na obrázku 2 lze vidět náhled do SQL Server Profileru s vloženým grafem čítačů. V grafu je použit pouze jeden čítač Procesor time, ale není problém do grafu zahrnout další nabízené čítače z široké nabídky.



Obrázek 2: SQL Server Profiler s grafem čítačů.

3 Popis vlastní aplikace

3.1 Podrobnější specifikace zadání

Cílem této práce je vytvořit aplikaci k analýze *souboru s vytížením* nad SQL Server databází určenou pro administrátory. *Soubor s vytížením* je do aplikace importován v podobě *XML souboru s vytížením* vyexportovaném v nástroji SQL Server Profiler nebo přímo v podobě *binárního souboru s vytížením*. Aplikace vyhledá SQL příkazy, které se liší pouze svými parametry. Vytvoří tak seznam unikátních příkazů. Parametry jsou ukládány do seznamů k unikátním příkazům. Pomocí grafického rozhraní si bude možno prohlédnout provoz v jednotlivých sezeních (session). Lze vykreslit graf zobrazující *návaznosti SQL příkazů* buď jednoho sezení nebo celého vytížení. V grafu bude ke každé hraně uveden počet opakování. Naimportovaný *soubor s vytížením* bude možno vyexportovat do externího souboru. Vyexportovaný soubor bude obsahovat unikátní SQL příkazy včetně jejich parametrů. Počet parametrů ve vyexportovaném *souboru návazností SQL příkazů* je možno procentuálně omezit. Tímto opatřením lze redukovat velikost vyexportovaného souboru. Ve vyexportovaném *souboru návazností SQL příkazů* zůstanou také zachovány časy spuštění SQL příkazů včetně sezení ve kterých byly spuštěny. Vyexportovaný *soubor návazností SQL příkazů* by měl tedy obsahovat veškeré informace pro znovuspuštění celého provozu. Po nalezení nedostatků v souboru s vytížením, lze provést opatření, speciální aplikací spustit provoz znovu, opět zachytit provoz a znovu analyzovat zda došlo ke zlepšení.

3.2 Vstupy

Vstupem aplikace je *soubor s vytížením*, které získáme (odchytneme) nad SQL Serverem. Aplikace pro import podporuje dva formáty - *binární soubor s vytížením* a *XML soubor s vytížením*. Import *binárního souboru s vytížením* však vyžaduje na daném počítači instalaci Sql Serveru, protože využívá některé knihovny, které jsou součástí Sql Serveru. Import *XML souboru s vytížením* je podporován vždy.

3.3 Výstupy

3.3.1 Vykreslení grafu návazností SQL příkazů

Možnosti nastavení zobrazování grafu v aplikaci:

- Podle množství dat:
  - Pouze pro konkrétní sezení.
    V případě, kdy uživatel aplikace (DBA) řeší problém na serveru a ví, že v konkrétním sezení nastává problém, který chce analyzovat, pak není potřeba vykreslovat graf všech sezení. V takovém případě by graf obsahoval spoustu nežádoucích dat a DBA proto může využít funkcionality pro vykreslení právě konkrétní sezení.

  - Pro všechny sezení.
    Zobrazení celého souboru s vytížením najednou.

- Podle vypsání příkazů.
  Na databázi můžou být zasílány krátké, ale i velice dlouhé příkazy, například s JOINy a vnořenými dotazy. Pokud by byl zobrazen graf, kde se nacházejí dlouhé příkazy, bylo by velice nepřehledné se v takovém grafu vyznat. Aplikace proto nabízí použití překladové tabulky. Do grafu se místo konkrétních dlouhých příkazů zapíšou čísla. V překladové tabulce lze podle těchto čísel dohledat celý příkaz.

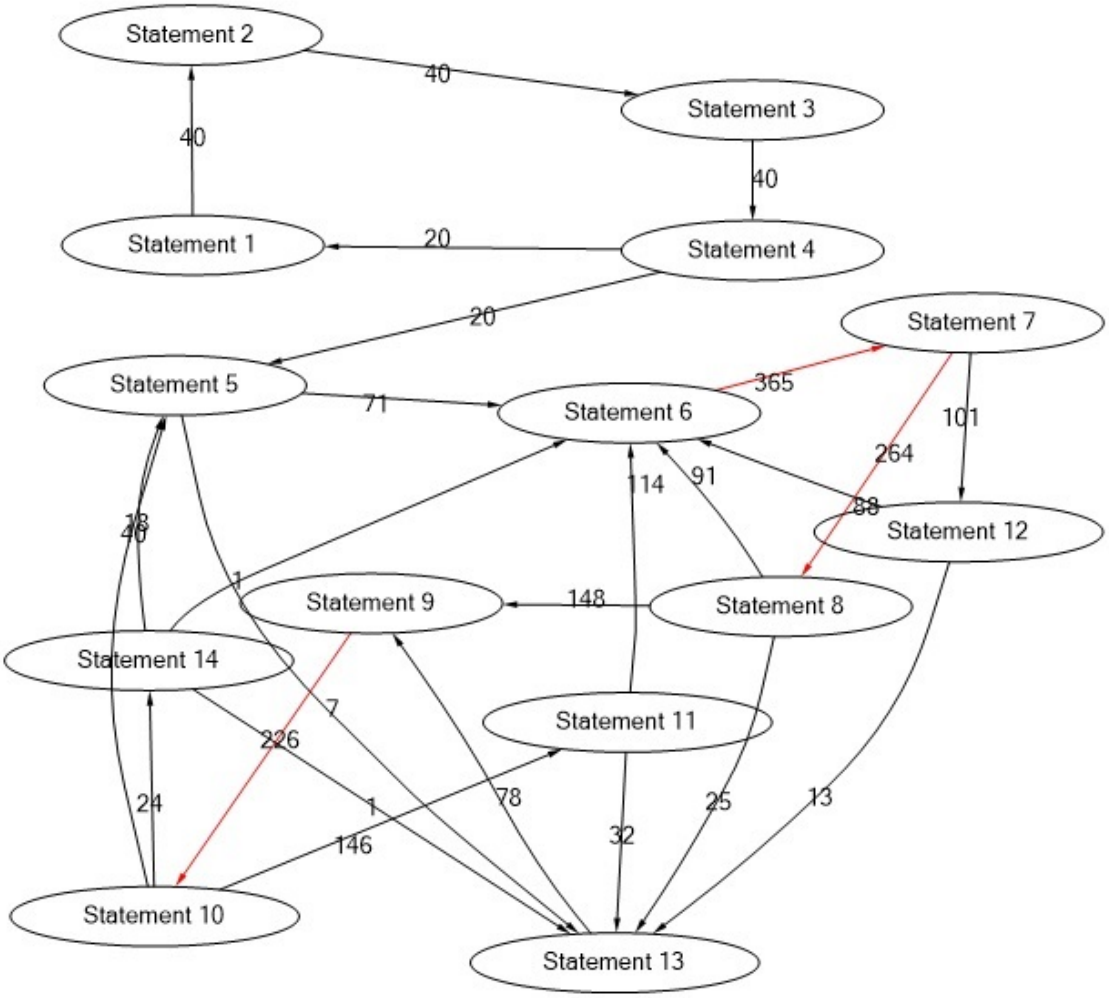
  - S použitím překladové tabulky
  - Bez použití překladové tabulky

- Zvýraznením hran mezi SQL příkazy podle počtu opakování.
  Pro větší *soubory s vytížením* se graf stává málo přehledným, lze proto nastavit zvýraznění často se opakujících hran grafu.

  Vykreslení *grafu návazností SQL příkazů* umožňuje:

  - **Zpětnou kontrolu UML diagramů.**
    Pokud máme k dispozici UML diagramy informačního systému, můžeme porovnat operace popsané v diagramech s provozem vykresleným v grafu. Operace v diagramu by se měly blížit operacím ve vykresleném grafu.
  - **Vygenerování velkého souboru s vytížením.**
    Tato funkcionalita je prozatím vize do budoucna. Z malého souboru s vytížením by aplikace mohla být schopna vygenerovat soubor s vytížením několikanásobně větší, při zachování poměru jednotlivých hran (přechodu z jenoho SQL příkazu na druhý). Takto vygenerovaný soubor s vytížením by se velice blížil původnímu vytížení nad databází a to za pomocí informace o počtu opakování daných hran mezi SQL příkazy.
  - **Adepti na uložené procedury/transakce.**
    Z grafu lze vyčíst často se opakující sekvence SQL příkazů a zamyslet se, zda by nebylo vhodné obalit danou sekvenci SQL příkazů do procedury (transakce), které bude uložená na databázovém serveru. Ne vždy je to možné, ale takové opatření může vést ke zvýšení výkonu databáze. Například při použití složitějšího formuláře, který ukládá do více tabulek se na databázový server odesílá více než jeden příkaz. Pokud by na straně serveru existovala uložená procedura (transakce), stačilo by jedno odeslání příkazu po síti. Do budoucna bych rád přidal funkcionalitu pro nabídnutí adeptů na uložené procedury-/transakce.

  Na obrázku 3 můžeme vidět vykreslení *grafu návazností SQL příkazů* s použitím překladové tabulky a zvýrazněním často se opakujících *návazností SQL příkazů* (hran grafu).



Obrázek 3: Vykreslení grafu návazností SQL příkazů

3.4 Funkce aplikace

3.4.1 Základní přehled

Po naimportování souboru s vytížením aplikace zobrazí session id (identifikátory sezení), které je možno rozkliknout a zobrazit výpis všech SQL příkazů zaslaných na databázi v rámci zvoleného sezení v pořadí v jakém byly na databázi zaslány. Pokud obsahuje SQL příkaz parametry, pak jsou jejich hodnoty nahrazeny otazníky.

Hodnoty parametrů však nejsou ztraceny. Po kliknutí na SQL příkaz se v aplikaci zobrazí všechny hodnoty parametrů, které byly stejným SQL příkazem na databázi v rámci zvoleného sezení zaslány.

3.4.2 Analýza SQL příkazů

Analýza SQL příkazů poskytuje jednoduchý náhled na vytížení ze dvou základních pohledů:

- Dlouho trvající SQL příkazy.
  - Seznam dlouho trvajících SQL příkazů nám umožní prvotní náhled na SQL příkazy, které by mohly být problémové a je potřeba jim věnovat zvýšené pozornosti.

- Unikátní SQL příkazy.
  - Výpis všech typů SQL příkazů zasílaných na databázi. Lze vyexportovat do JSON souboru a využít seznam SQL příkazů pro ladění výkonu databáze.

3.4.3 Export souboru návázností SQL příkazů

Pří vývoji informačních systémů je potřeba již ze začátku udělat dobrý fyzický návrh databáze, aby byl schopen zvládat velký provoz. Takový fyzický návrh vychází z odhadovaného budoucího provozu a vždy se liší od provozu realného. Po zaběhnutí IS je tedy na místě upravit fyzický návrh podle reálného provozu a právě tomu nám pomůže vytvořená aplikace.

Export návazností SQL příkazů do *XML souboru návazností* je jedna z důležitějších funkcí celé aplikace. Pokud by jsme aplikaci rozšířili o podporu Oraclu či jiných SŘDB pak by nám aplikace sloužila jako univerzální nástroj pro převedení souboru s vytížením více SŘDB do jednotného formátu. Soubor s vytížením v tomto formátu by bylo možné znovu spouštět v jednotném nástroji.

Soubor návazností SQL příkazů může být použit pro vícevlaknové znovuspuštění vytížení nad databází pomocí jednoduché aplikace. Po odladění databáze můžeme znovu spustit celý soubor s vytížením a zkoumat zda došlo ke zlepšení. Tím pádem bude k dispozici jednotný pomocník pro ladění databáze a to pro Oracle, Sql Server a případné další SŘDB. Jelikož se časem mění požadavky na IS a dochází k dalšímu vývoji, není aplikace rozhodně na jedno použití. Aplikace pro spuštění *návazností SQL příkazů* není součástí této práce.

3.5 Návrh

3.5.1 Logování chyb

Nejdříve bych rád popsal sice vedlejší funkci, ale za to využívanou napříč celou aplikací a tou je logování. Logování pomáha odhalit mnohé chybné situace, ke kterým může v aplikaci docházet. Pro logování slouží metoda *Log* třídy *Debugger* náležící do namespace *Utilities*. Zprávy jsou logovány do textového souboru v následujícím formátu:
typ logované zprávy : [datum a čas] třída -> metoda kde bylo logovani vyvoláno: zpáva

Například:
Failure: [10. 3. 2015 22:43:40] : XmlFile->Import: Soubor je ve špatném formátu.

Rozlišují se tyto typy logovacích zpráv:

- Info - Informativní zpráva. Například: Počet naimportovaných SQL příkazů, celkový čas importu aj.
- Warning - Varovná zpráva, aplikace běží bez vážných následků, ale problém by měl být opraven.
- Fault - Stav, kdy aplikace neplní požadovanou funkci, aplikace běží dál ale může způsobit vážné následky.
- Error - Stav kdy se liší skutečný výstup od očekávaného.
- Failure - Stav kdy program není schopen provést požadovanou funkcionalitu.

Log zprávy jsou ukládány do souboru log.txt v adresáři exe souboru. Log soubor nám může pomoct efektivně odhalit a odstranit případné potíže aplikace. Soubor lze také zaslat vývojáři aplikace k opravení chyb. Log soubor obsahuje, všechny důležité informace daného problému.

3.6 Vykreslení grafu návazností SQL příkazů

K vykreslení grafu využívám knihovnu společnosti Microsoft s názvem *Microsoft Automatic Graph Layout*, dále jen *MSAGL*. Při výběru knihovny jsem vyzkoušel i jiné knihovny jako jsou: *Graph#, QuickGraph, GraphViz a jiné*. Ačkoliv knihovny vypadaly na první pohled velmi dobře, tak se mně nepodařilo knihovnu přízpůsobit vyhovujícím požadavkům. V některých případech se mně ani nepodařilo knihovnu zprovoznit. Při přechodu na knihovnu MSAGL se nenaskytl žádný problém a vykreslování grafů okamžitě fungovalo.

3.6.1 Uložiště

Před začátkem importu *souboru s vytížením* lze nastavit uložiště pro importovaná data. Výchozí uložiště je v paměťi RAM. Uložiště na HDD je nabízeno pro případ importu velkých souborů v řádů GB, které by zaplnily celou paměť RAM. Na výběr tedy jsou dvě uložiště:

- HDD - Na disku je vytvořena Sqlite databáze a následně je plněna daty.

  Výhody: Nízké nároky na paměť RAM.

  Nevýhody: Pomalá práce s daty.

- RAM - Veškeré data jsou uloženy v hlavní paměti. Tato možnost je zvolena defaultně.

  Výhody: Rychlejší práce s daty.

  Nevýhody: Vysoké nároky na paměť RAM.

3.6.2 Import

U importovaného *souboru s vytížením*, lze při zachytávání odfiltrovat SQL příkazy a také atributy. Důležité je aby *soubor s vytížením* obsahoval všechny povinné atributy.

Povinné atributy se kterými aplikace pracuje:

- TextData - SQL příkaz zaslaný na databázový server.
- SPID - Identifikátor sezení.
- Duration - Doba vykonávání celého procesu na databázovém serveru.
- StartTime - Čas počátku vykonávání SQL příkazu.
- EndTime - Čas dokončení vykonávání SQL příkazu.

Kromě těchto povinných atributů je zbytečná přítomnost atributů jiných, aplikace pracuje pouze s povinnými atributy a všechny ostatní atributy pouze zpomalují proces importu.
Plánována podpora atributů do budoucna:

- CPU - Procesorový čas.
- Reads - Logické čtení z disku.
- Čítače z nástroje Performance Monitor.

Pokud by aplikace zpracovávala v budoucnu i plánované atributy, byla by schopna nabídnout mnohem širší analýzu včetně různých nápověd, respektive tipů pro uživatele. Tyto tipy by směřovaly k lepší konfiguraci, či lepšímu fyzickému návrhu.

Import souboru s vytížení lze provést ve dvou formátech:

- Binární soubor s vytížením - Tento soubor je v uložen v binární podobě a nelze ho přečíst jako klasický textový soubor. Pro čtení *binárního souboru s vytížením* využívá aplikace třídu **TraceFile** náležící do namespace *Microsoft.SqlServer.Management.Trace*. Důležité je zmínit, že import tohoto formátu je podporován pouze pokud je na zařízení nainstalován SQL Server.

- XML soubor s vytížením - Pro čtení využívá aplikace standardní třídu XmlReader, která je součástí namespace *System.Xml*.

3.6.3 Parsování SQL příkazů.

Jádro celé aplikace tvoří parsování SQL příkazů k němuž aplikaci napomáhá externí knihovna s názvem SqlParser (http://www.sqlparser.com). Výhodou je, že tato knihovna podporuje také ostatní databázové systémy. Případné rozšíření funkčnosti o podporu databázového systému jako je Oracle, jsou dveře otevřené. SqlParser umí každý SQL příkaz rozložit do specialní stromové struktury a umožňuje s takto rozloženým příkazem pracovat. Na oficiálním webu parseru [7] se nachází příklady stromových struktur pro různé typy SQL příkazů. Pro základní SQL příkazy, jako jsou Select, Delete, Insert, Update a CreateTable, je stromová struktura jednotná nezávisle na SŘDB. Za předpokladu, že by bylo dostačující, kdyby aplikace zpracovávala pouze tyto základní SQL příkazy a zároveň by byl k dispozici *soubor s vytížením* ve stejném formátu pro každý SŘDB, pak by se rozšíření aplikace o podporu dalších SŘDB obešlo naprosto bez zásahu.

Kromě základních SQL příkazů existují také SQL příkazy, které se na každém SŘDB liší. V případě SQL Serveru se jedná například o SQL příkazy typu: declare, execute, set, aj. Pro tyto případy je nutno upravit implementaci práce se stromem pro každý druh SQL příkazu. Pomocí vytvořené stromové struktury lze cyklem projít klauzule SQL příkazů, jako jsou SET, WHERE, ORDER BY apod. a vybrat z nich informace, které nás zajímají. Způsob parsování použité v aplikaci lze rozdělit do několika fází:

- **Kontrola validity SQL příkazu.**
  Téměř všechny SQL příkazy zaslané na DB jsou generovány aplikací určenou pro obsluhu nějakého informačního systému (IS) za kterou stojí uživatel. Aplikace by měla být řádně otestována a nemělo by se stát, že aplikace bude generovat nevalidní SQL příkazy. Výjimku by mohli tvořit administrátoři, kteří můžou ručně psát SQL příkazy a zasílat je na databází, kdy se může lehce stát, že se přepíšou a zašlou na DB nevalidní SQL příkaz. Tuto situaci však nebudeme brát v úvahu. Zajímá nás provoz uživatelů, tedy SQL příkazy generované aplikacemi, nikoliv provoz administrátorů. Provoz administrátorů by se nám do našeho *souboru s vytížením* neměl plést. Pokud aplikace nalezne nevalidní SQL příkaz pak je do logu vložen varovný záznam, SQL příkaz se zahodí a již se s ním dále nepracuje. Do budoucna by bylo vhodné tyto SQL příkazy ukládat a umožnit uživateli si v aplikaci zobrazit nevalidní SQL příkazy.

- **Rozpoznání typu SQL příkazu.**
  SqlParser umožňuje rozpoznávání SQL příkazů díky němuž lze jednoduše SQL příkazy třídit do jednotlivých kategorií. Aplikace rozlišuje následující typy SQL příkazů:

  - **DML (Data Manipulation Language)** - Jazyk pro manipulaci s daty. Update, Delete, Insert.

- **DDL (Data Definition Language)** - Jazyk pro definici dat: Create, Alter, Drop
- **DQL (Data Query Language)** - Jazyk pro dotazování: Select
- **TCL (Transaction Control Language)** Jazyk pro řízení transakcí - : Commit, Rollback, Set
- **EXEC** - Zejména volání procedur volaných pomocí příkazu exec, execute, sp_prepexex

• **Rozložení SQL příkazu.**
SqlParser rozloží celý SQL příkaz do stromové struktury, čímž umožní procházení jednotlivých elementů SQL příkazů v cyklu.

• **Přečtení hodnot atributů a nahrazení.**
Aplikace projde algoritmem stromovou strukturu SQL příkazu a nahradí hodnoty atributů otázníky. Původní hodnoty jsou uloženy do kolekce. Výpis třech ukázkových vstupních SQL příkazů je uveden ve výpisu kódu 1.

```
SELECT id, email FROM user WHERE firstname = 'Karel' AND lastname = 'Novak'

UPDATE user SET email = 'karel.novak@email.com', phone = '111222333' WHERE id = 1

DECLARE @p1 int
SET @p1=1
exec sp_prepexec @p1 output,N'@P0 int',N'SELECT * FROM uzivatel WHERE id =
    @P0',171768
SELECT @p1
```

Výpis 1: Vstupní SQL příkazy (před parsováním)

Vstupní SQL příkazy jsou aplikací zpracovány a jejich zobrazení pro uživatele aplikace je ukázáno ve Výpisu 2.

```
SELECT * FROM user WHERE firstname = ? AND lastname = ?

UPDATE user SET email = ?, phone = ? WHERE id = ?

SELECT * FROM uzivatel WHERE id = ?
```

Výpis 2: Transformované SQL příkazy parserem

K takto transformovaným SQL příkazům se uchovají parametry v kolekci a budou vypadat následovně:

1. 'Karel', 'Novak'
2. 'karel.novak@email.com', '111222333', 1
3. 171768

V případě třetího příkazu si také aplikace zapamatuje, že je SQL příkaz zaslán parametrizovaně. Tato informace se projeví v exportovaném souboru.

Poté co každý SQL příkaz projde přes všechny zmíněné fáze je vrácena instance třídy Command se všemi potřebnými informacemi. Na obrázku 4 se nachází zjednodušený třídní diagram hlavní části aplikace včetně zmíněné třídy Command. Třída MainForm představuje okno grafického uživatelského rozhraní (GUI). Při volbě uložiště v GUI je informace o uložišti uchována v třídě MainController. Třída MainController obsahuje všechny potřebné metody pro práci s SQL příkazy. Na základě zvoleného uložiště či formátu, jsou volány metody příslušné třídy. V případě uložiště RAM jsou veškeré SQL příkazy uloženy přímo v třídě MainController v podobě instance třídy SqlCommands. V případě nastaveného uložiště na HDD se pracuje s instancí třídy SqLite. Každý importovaný příkaz musí projít procedurou pro parsování v třídě SqlParser.

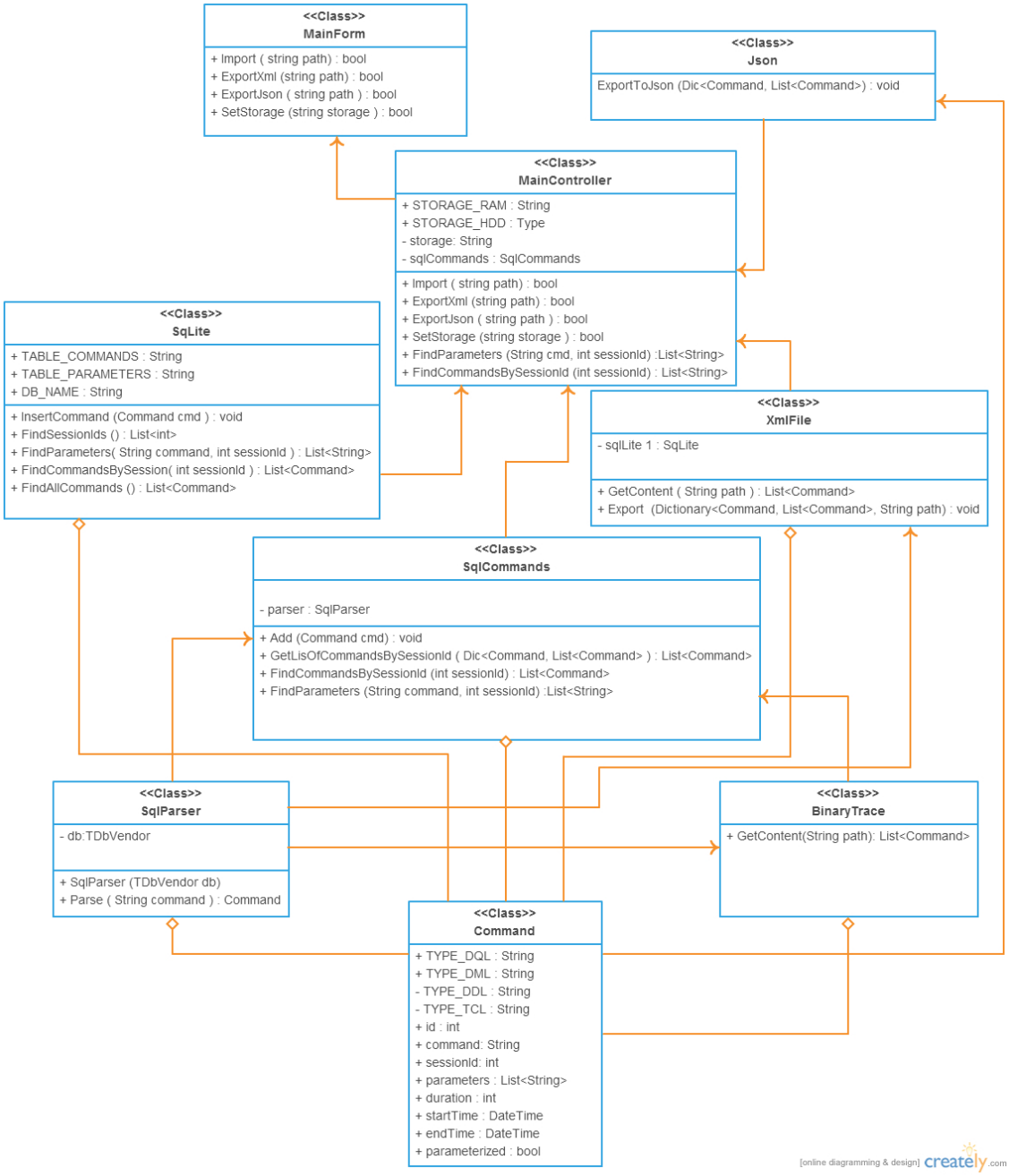
Na databázi je zasíláno více typů SQL příkazů a nelze se ke všem chovat stejně z pohledu parsování. Zejména v této fázi se můžou vyskytnout SQL příkazy, na které ještě aplikace není připravená a transformuje SQL příkaz nežadoucím způsobem. A právě tady se asi nejvíce může hodit dříve zmíněné logování. Vzhledem k velkému rozsahu SQL příkazů si ukážeme pouze jeden delší SQL příkaz pocházející z reálného vytížení, ostatní příkazy jsou úmyslně kratšího rozsahu.

Ukážeme si tedy několik typů SQL příkazů se kterými si aplikace musí poradit.

Příklad: Ve výpisu 3 je uveden SQL dotaz volán procedurou sp_executesql.

```
exec sp_executesql N' SELECT O_YEAR, SUM(CASE WHEN NATION = @m1
            THEN VOLUME
            ELSE 0
            END) / SUM(VOLUME) AS MKT_SHARE
   FROM (  SELECT datepart(yy,O_ORDERDATE)  AS O_YEAR,
            L_EXTENDEDPRICE * (1-L_DISCOUNT) AS VOLUME,
            N2.N_NAME           AS NATION
         FROM PART,
            SUPPLIER,
            LINEITEM,
            ORDERS,
            CUSTOMER,
            NATION N1,
            NATION N2,
            REGION
         WHERE P_PARTKEY = L_PARTKEY AND
            S_SUPPKEY = L_SUPPKEY AND
            L_ORDERKEY = O_ORDERKEY AND
            O_CUSTKEY = C_CUSTKEY AND
            C_NATIONKEY = N1.N_NATIONKEY AND
            N1.N_REGIONKEY = R_REGIONKEY AND
            R_NAME    = @m2 AND
            S_NATIONKEY = N2.N_NATIONKEY AND
            O_ORDERDATE BETWEEN ''1995-01-01'' AND ''1996-12-31'' AND
            P_TYPE    = @m3
      ) AS ALL_NATIONS
   GROUP BY O_YEAR
   ORDER BY O_YEAR',N'@m1 varchar(25),@m2 varchar(25),@m3 varchar(25)',@m1='
       MOROCCO',@m2='AMERICA',@m3='STANDARD ANODIZED BRASS';
```

Výpis 3: SQL dotaz volán procedurou sp_executesql



Obrázek 4: Třídní diagram aplikace

Příklad: Výpis 4 obsahuje parametrizovaný SQL příkaz pro vložení záznamu do tabulky. Tento příkaz je obalen funkcí sp_executesql.

```
exec sp_executesql N'
INSERT INTO ORDERS (O_ORDERKEY, O_CUSTKEY, O_ORDERSTATUS, O_TOTALPRICE,
                    O_ORDERDATE, O_ORDERPRIORITY, O_CLERK, O_SHIPPRIORITY,
                        O_COMMENT)
            VALUES
               (@m1, @m2, @m3, @m4, @m5, @m6, @m7, @m8, @m9)',N'@m1 int,@m2 int,@m3
                    char(1),@m4 decimal(15,2),@m5 varchar(25),@m6 char(15),@m7 char(15),@m8
                    int,@m9 varchar(79)',@m1=11724,@m2=83359,@m3='F',@m4=181610.54,@m5
                    ='1993-03-07',@m6='4-NOT SPECIFIED',@m7='Clerk#000000123',@m8=0,
                    @m9='nts. regular dolphins along the qu'
```

Výpis 4: SQL insert zaslaný parametrizovaně

Příklad: Ve výpisu 5 je ukázka Select dotazu obaleného funkcí sp_prepexec. Z pohledu parsování je potřeba přistupovat k tomuto příkazu jiným způsobem než tomu bylo u SQL příkazu ve výpisu 4.

```
declare @p1 int
set @p1=1
exec sp_prepexec @p1 output,N'@P0 int',N'Select * from uzivatel where id = @P0      ',90961
select @p1
```

Výpis 5: SQL dotaz volaný procedurou sp_prepexec

Příklad: Ve výpisu 6 je příklad jednoho SQL příkazu ze skupiny TCL :

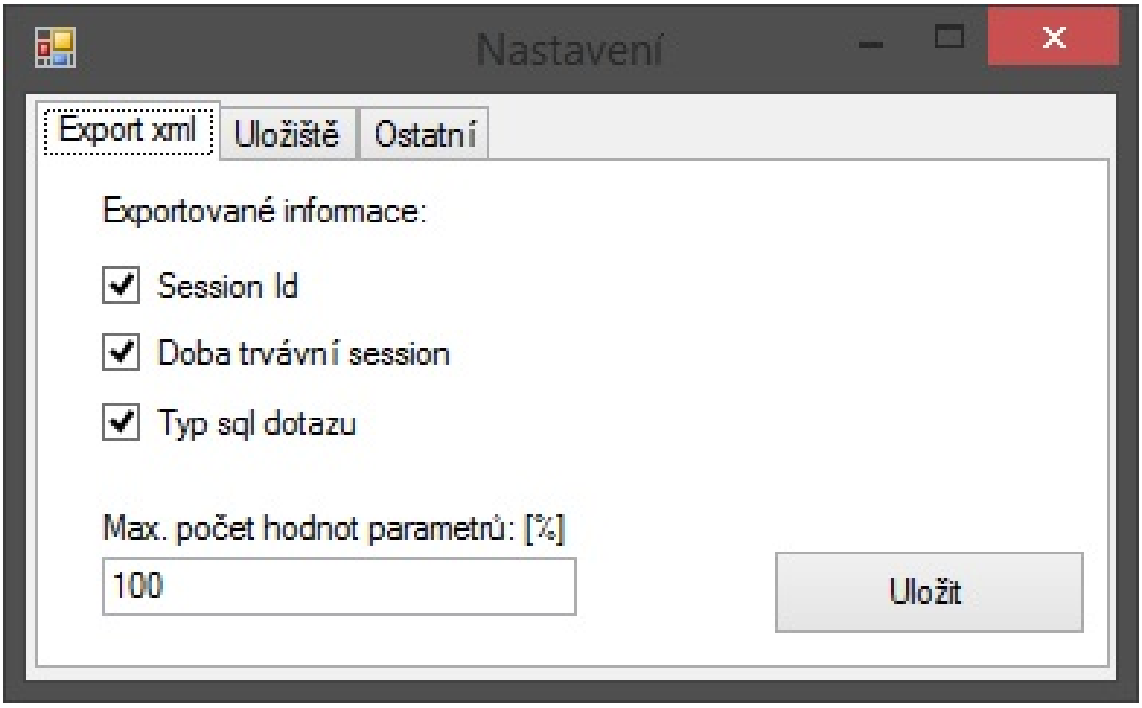
```
set transaction isolation level  read committed set  implicit_transactions  off
```

Výpis 6: SQL set transaction

3.6.4 Jednoduchá analýza

Aplikace nabízí zobrazení jednoduché analýzy. První pohled se týká zobrazení dlouho trvajících SQL příkazů. Tyto SQL příkazy jsou vybírány podle hodnoty atributu *Duration* neboli čas běhu procesu. Seznam je seřazen od nejhoršího (časově nejdéle běžícího) SQL příkazu po nejlepší. Druhý pohled umožňuje zobrazení všech druhů SQL příkazů, které se v *souboru s vytížením* vyskytují. V tomto seznamu se nevyskytují duplicity. Stejný SQL příkaz s různými hodnotami je chápán jako jeden.

Rozšířením podpory importovaných atributů ze souboru s vytížením, by mohla analýza poskytovat mnohem větší možnosti. S větším množstvím informací lze lépe pracovat a odvodit mnohem více závěrů. Uživateli by mohly být nabízeny konkrétní změny, které zvýší výkon databáze.



Obrázek 5: Nastavení exportu aplikace

3.6.5 Export

Soubor s vytížením naimportovaný do aplikace, lze také vyexportovat jako *soubor návazností SQL příkazů* ve formátu, který umožuje snadné spouštění *souboru s vytížením*. Na výběr je JSON nebo XML soubor návazností SQL příkazů.

Jelikož všechny defaultně exportované informace nejsou vždy potřebné nabízí aplikace v nastavení několik věcí, které lze ovlivnit. Na obrázku 5 můžeme vidět nastavení exportu v aplikaci.

V případě, že není potřeba v exportu přítomnost atributů jako jsou *session ID*, doba trvání sezení nebo typ SQL příkazu pak jej stačí odškrtnout a v exportu nebudou. Tím dochází k šetření místa.

Avšak významnější je nastavení maximálního množství parametrů vyjádřeno v procentech. Defaultní hodnota je 100%. Pokud by vyexportovaný soubor s vytížením obsahoval často se opakující komplexní SQL příkaz obsahující, například 10 a více parametrů, pak by velikost exportovaného souboru velice rychle rostla a přitom nás hodnoty parametrů nemusejí vůbec zajímat. Nanejvýš může postačit několik málo procent ukázkových hodnot. S ponecháním všech hodnot roste velikost souboru.

Popis formátu vyexportovaného *XML souboru návazností SQL příkazů*:

- **SQL příkazy** Obsahují atributy:
  - **Id** - Identifikátor SQL příkazu v rámci xml souboru.

- **Text** - Sql příkaz.
- **Paramcount** - Počet parametrů SQL příkazu.
- **Count** - Počet opakování SQL příkazu.
- **Type** - Typ SQL příkazu (DQL, DML, DDL, TCL, EXEC).
- **Parameterized** - Počet SQL příkazů zaslaných parametrizovaně.
- **Unparameterized** - Počet SQL příkazů zaslaných neparametrizovaně.

V případě, že SQL příkaz obsahuje parametry pak se zde hodnoty parametrů vypíšou. Jak již bylo výše zmíněno, výpis hodnot parametrů je možno omezit v nastavení.

- **Sezení** obsahuje:

  Atributy:

  - **StartTime** - Začátek sezení.
  - **EndTime** - Čas ukončení sezení.
  - **Duration** - Doba běhu sezení.
  - **SessionId** - Identifikátor sezení.

  SQL příkaz:

  - **Id** - Identifikátor volaného SQL příkazu.
  - **Param** - Parametry SQL příkazu.
  - **Startat** - Čas započetí sezení.

  Poznámka 3.1 Co se časů týče nejedná se o skutečné časy, kdy sezení započalo či skončilo. Časy jsou relativní vzhledem k započnutí sledování. První čas začíná v nula hodin.

```
<Workload count="14">
  <sql id="1" text="set implicit_transactions on " paramcount="0" count="40"
  type="TCL" parameterized="0" unparameterized="40"/>
  <sql id="2" text="set transaction isolation level serializable " paramcount="0"
  count="40" type="TCL" parameterized="0" unparameterized="40"/>
  <sql id="3" text="Select * From uzivatel Where id = ?" paramcount="1" count="78"
  type="DQL" parameterized="78" unparameterized="0">
    <values count="1">
      <p>171768</p>
    </values>
    <values count="1">
      <p>132035</p>
    </values>
    <values count="1">
      <p>67263</p>
    </values>
  </sql>
  <sql id="4" text="Select * From podkategorie Where kategorieId = ?" paramcount="1"
  count="365" type="DQL" parameterized="365" unparameterized="0">
    <values count="60">
      <p>2</p>
    </values>
  </sql>
  <Session startTime="0:00:00.000" endTime="0:03:36.703" duration="00:03:36.7030000"
  sessionId="59">
    <statement id="1" param="0" startat="0:00:00.000"/>
    <statement id="2" param="0" startat="0:00:00.017"/>
    <statement id="3" param="0" startat="0:00:00.017"/>
    <statement id="4" param="0" startat="0:00:00.033"/>
    <statement id="4" param="0" startat="0:00:00.097"/>
    <statement id="4" param="0" startat="0:00:00.097"/>
    <statement id="4" param="0" startat="0:00:00.097"/>
  </Session>
```

Obrázek 6: Ukázka vyexortovaného XML souboru

3.7 Testování aplikace

Testování aplikace je zaměřeno na problémovou část, kterou se jeví Import *souboru s vytížením*. Testovány byly celkem tři *soubory s vytížením* s uložištěm na HDD i RAM. V tabulkách 1, 2, 3 jsou výsledky testování importu *souboru s vytížením*. Testování importu se zvoleným uložištěm na HDD dopadlo výrazně hůře než pro uložiště v paměti RAM. Horší výsledky si lze vysvětlit několika pohledy.

Zatímco přístupová doba paměti RAM se pohybuje v ns, přístupová doba HDD se pohybuje v ms, což se nepříznivě projevuje na běhu aplikace při importu s využitím uložiště na HDD. Budeme-li uvažovat situaci importu souboru s vytížením o velikosti 662 MB a celkovým počtem 14 637 215 SQL příkazů, pak se bude do paměti přistupovat právě 14 637 215 krát. Vkládání také zpožduje select dotazy, které je nutné na databázi pokládat při importu souboru s vytížním do databáze. Je potřeba omezit počet přístupů na disk čehož lze docílit hromadným vkládáním (multiple-row INSERT).

Samotná doba přístupu však zajisté není jediným aspektem ovlivnující výkon databáze SqLite. Bylo by vhodné zabývat se možnostmi nastavení jako je například velikost vyrovnávací paměti. Konfigurací SqLite jsem se v této práci nezabýval a využívám výchozí nastavení.

```
INSERT INTO commands (id, command, type, sessionId, duration, startTime, endTime,
    parameterized) VAUES
(NULL, @command1, @type1, @sessionId1, @duration1, @startTime1, @endTime1,
    @parameterized1),
(NULL, @command2, @type2, @sessionId2, @duration2, @startTime2, @endTime2,
    @parameterized2),
(NULL, @command3, @type3, @sessionId3, @duration3, @startTime3, @endTime3,
    @parameterized3),
(NULL, @command4, @type4, @sessionId4, @duration4, @startTime4, @endTime4,
    @parameterized4)
```

Výpis 7: Hromadné vkládání záznamů

Z výpisu kódu 7 je patrné, že aplikace využívá parametrizované hromadné vkládání. Jak je uvedeno na oficiálním webu SQLite [4], existuje konstanta, která omezuje počet těchto parametrů v SQL příkazu na 999 a hodnotu nelze změnit. Tato konstanta se nazývá `SQLITE_MAX_VARIABLE_NUMBER`.

Kvůli tomuto omezení by bylo výhodnější použít programátorsky nečistý způsob a to neparametrizované SQL příkazy. Parametrizovaně můžeme sestavit tento hromadný insert maximálně pro circa 140 vložení najednou, kdežto u neparametrizovaného lze využít většího počtu vložených záznamů najednou. Neparametrizované vkládání aplikace neumožňuje z důvodu narušení SQL příkazu SQL injection. Takto narušený SQL příkaz by vyvolal výjimku a aplikace by zobrazila chybovou hlášku. Pokud by aplikace měla používat neparemetrizované vkládání, byla by potřeba prozkoumat možnosti, kterými by při spojování řetězců mohl být SQL příkaz narušen a vytvořit funkce, které by umožnily neparametrizovaný hromadný insert a zaručoval validní výstupy.

Vytížení	Velikost [MB]	Počet SQL příkazů
Velké	662	14 637 215
Střední	54	365 773
Malé	6	114 539

Tabulka 1: Přehled testovaných souborů s vytížením

Vytížení	Čas importu [min:sec]	Zpracovaných SQL příkazů za sekundu
Velké	04:30.00	54 211
Střední	0:30.28	12 079
Malé	00:02.37	48 328

Tabulka 2: Zátěžové testování importu dat s uložištěm na RAM

Na příkladu si ukážeme jak chyba vznikne. Vezměme v úvahu jednoduchý insert pro vložení právě jednoho záznamu. Nejprve si ukážeme jak vypadá SQL příkaz, který proběhne úspěšně.
Vstupní hodnoty do aplikace:

◦ **Id** - NULL

◦ **Command** - select * from user

◦ **Type** - DQL

◦ **SessionId** - 15

◦ **Duration** - 20

◦ **StartTime** - 0:00:000

◦ **EndTime** - 0:00:000

◦ **Paremeterized** - 0

```
INSERT INTO commands (id, command, type, sessionId, duration, startTime, endTime,
    parameterized) VAUES
(NULL, 'select * from user', 'DQL', '15', '20', '0:00:000', '0:00:000', 0);
```

Výpis 8: Ukázka jednoduchého insertu

Vytížení	Čas importu [min:sec]	Zpracovaných SQL příkazů za sekundu
Velké	10:26.00 - 12:34:00	19 412 - 22 978
Střední	00:38.92	9 398
Malé	0:06.9	16 599

Tabulka 3: Zátěžové testování importu dat s uložištěm na HDD

Velikost bufferu	Doba Importu
20	18:06.99
60	16:00.13
100	15:31.42
140	11:59.84

Tabulka 4: Velikost bufferu vs. doba importu

Pokud jen trochu poupravíme SQL příkaz v atributu command a ostatní hodnoty ponecháme pak Insert narušíme.

Vstupní hodnoty ponecháme stejné s výjimkou atributu *Command*.

○ **Command - select * from user where email = 'franta.novak@email.cz'.**

Pro lepší přehlednost oddělíme jednotlivé hodnoty vždy na nový řádek. Viz výpis 9

```
INSERT INTO commands (id, command, type, sessionId, duration, startTime, endTime,
    parameterized) VAUES
(
NULL,
'select * from user where email = '
franta.novak@email.cz'',
'DQL', '15', '20',
'0:00:000',
'0:00:000',
0
);
```

Výpis 9: Ukázka jednoduchého insertu narušeného dírou SQL Injection

Na modifikovaném kódu v úkázce 9 si můžeme všimnout, že nám apostof předčasně ukončil hodnotu atributu, kterou chceme vložit. Kompilátor by poté očekával čárku, která bude odělovat hodnoty, tu již nenalezne a následně kompilátor nalezne řetězec bez uvozovek (apostrofů). V takovém to případě se nám příkaz pro vložení záznamu narušil, není korektní a nemůže být vykonán, v aplikaci nastane chyba vyvolaná knihovnou SqlParser ve fázi kontroly validity SQL příkazu.

Provedl jsem testování jaký vliv má velikost bufferu aplikace pro hromadné vkládání do databáze na dobu importu. V tabulce 4 je porovnání časů při různých velikostech bufferu.

Další aspekt ovlivňující rychlost ukládání na disk jsou různé služby Windows. Při zapisování změn na disk, spouští Windows služby, které mapují změny. Služby potřebují přistupovat k disku, kde zjišťují jaké nastaly změny, což opět zhoršuje dobu přístupu.

4 Závěr

Cílem práce bylo vytvořit nástroj umožňující jednoduchou analýzu provozu SQL Serveru. Tento cíl se podařilo naplnit v jednoduché aplikaci nabízející základní přehled nad *souborem s vytížením*.

První bod práce bylo nalezení SQL příkazů, které se lišší pouze svými parametry. Tohoto cíle se mně podařilo dosáhnout v části Parsování SQL příkazů 3.6.3, kde je detailně popsán průběh procesu parsování. Druhý bod, oddělení hodnot parametrů SQL příkazů do jednotlivých seznamů, přímo souvisí s bodem prvním a je součástí celého procesu parsování. Následující třetí bod, vytvoření grafu *návazností SQL příkazů* je popsán v části Vykreslení *grafu návazností SQL příkazů* 3.3.1. Uložení neboli export je popsán v kapitole 3.6.5. Export odpovídá splnění čtvrtého bodu zadání. Umožňuje jak požadovaný JSON formát, tak XML formát. Poslední bod, tedy testování je důkladně probráno v poslední části Testování aplikace 3.7.

Během vývoje aplikace jsem dostal příležitost nahlédnout hlouběji do SQL Serveru z několika hledisek. Získal jsem větší přehled nad SQL příkazy, které jsou na server zasílány a následně jsem s nimi pracoval. Naučil jsem se používat některé nástroje SQL Serveru, které využívám také pro spolupráci s vytvořenou aplikaci. Získal jsem zkušenost s používáním externích knihoven.

Jak již bylo zmíněno, aplikace zatím nabízí pouze základní analýzu a nelze s ní plnohodnotně pracovat jako s hotovými nástroji k tomuto účelu sloužícími. Do budoucna by však bylo velkou výhodou do aplikace integrovat přehrávání souboru s vytížením. Nyní aplikace umožnuje pouze vyexportovat *soubor s vytížením*. Vyexportované vytížení je možno využít pomocí speciální aplikace, které je schopná provoz spustit opakovaně. Tato aplikace však není součást této práce. Uživatel by jistě uvítal funkcionalitu přehrávání vytížení přímo v této aplikaci. Současně se znovu přehráváním vytížení by se dala relativně snadno implementovat funkcionalita zachytávání vytížení, alespoň co se týče Sql Serveru. Uživatel by tedy nebyl nucen zachytávat *soubor s vytížením* v jiných nástrojích a následně ho importovat do aplikace. Také rozšíření aplikace o různé čítače z Performance Monitoru by poskytla mnohem více informací, které by se daly využít. Aplikace by mohla uživateli sama doporučovat změny a to jak změny týkající se konfigurace SQL Serveru, tak změny týkající se fyzického návrhu databáze. Tím by se dalo docílit minimalizace nároků na uživatele v oblasti znalostí SQL Serveru a databází. Se zmíněnými rozšířeními by již aplikace mohla nabízet testování, kdy by znovu přehrála stejný *soubor s vytížením* a vyhodnotila zda došlo k celkovému zlepšení.

Před nasazením do ostrého provozu by bylo nutné vyřešit problematiku licencí knihoven, které aplikace využívá. Jedná se o knihovnu SqlParser, knihovnu pro vykreslování grafů Microsoft Automatic Graph Layout a databázový systém SQLite. Jak vyplývá z testování aplikace, bylo by vhodné zapracovat na technologii ukládání na disk do databáze SqLite, která výrazně zpomaluje běh aplikace.

Ondřej Krajčík

5 Reference

[1] R. STANEK, William. *SQL Server 2012: Kapesní rádce administrátora*. 2013. vyd. Brno: Computer Press, 2013. ISBN 978-80-251-3797-0.

[2] *MS SQL: Top 10 SQL Server Counters for Monitoring SQL Server Performance*. A. LARSEN, Gregory. Database Journal: The Knowledge Center for Database Professionals [online]. 2011 [cit. 2015-04-18]. Dostupné z: http://www.databasejournal.com/features/mssql/article.php/3932406/Top-10-SQL-Server-Counters-for-Monitoring-SQL-Server-Performance.htm

[3] *CO TAKOVÝ DATABÁZOVÝ ADMINISTRÁTOR DĚLÁ?*. SOLAŘ, Tomáš. Tomáš Solař [online]. [cit. 2015-04-18]. Dostupné z: http://www.tomas-solar.com/co-takovy-databazovy-administrator-dela/

[4] *Limits In SQLite. SQLITE. SQLite* [online]. 2015 [cit. 2015-04-18]. Dostupné z: https://www.sqlite.org/limits.html

[5] *DB-Engines Ranking - Trend of Relational DBMS Popularity.* DB-ENGINES: Knowledge Base of Relational and NoSQL Database Management Systems [online]. [cit. 2015-04-25]. Dostupné z: http://db-engines.com/en/ranking_trend/relational+dbms

[6] BAČA, Radim. *Praktický manuál k administraci SQL Serveru*. 23. 2. 2015. 2015, 36 s.

[7] *SQL query parse tree in XML*. General SQL Parser: Profesional SQL engine for various databases [online]. 2015 [cit. 2015-04-26]. Dostupné z: http://www.sqlparser.com/sql-query-parse-tree-in-xml.php